04 | Tensor：PyTorch中最基础的计算单元

2021-10-18 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远
时长 16:28 大小 15.09M

在上节课中，我们一起学习了 NumPy 的主要使用方法和技巧，有了 NumPy 我们可以很好地处理各种类型的数据。而在深度学习中，数据的组织则更进一步，从数据的组织，到模型内部的参数，都是通过一种叫做**张量**的数据结构进行表示和处理。

今天我们就来一块儿了解一下张量（Tensor），学习一下 Tensor 的常用操作。

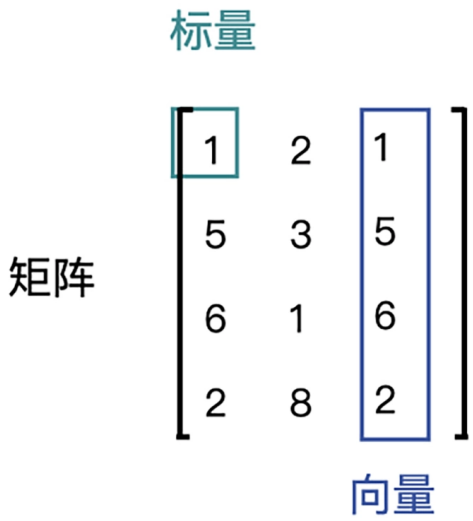
什么是 Tensor

Tensor 是深度学习框架中极为基础的概念，也是 PyTroch、TensorFlow 中最重要的点之一，它是一种数据的存储和处理结构。

回忆一下我们目前知道的几种数据表示：

1. 标量，也称 Scalar，是一个只有大小，没有方向的量，比如 1.8、e、10 等。
2. 向量，也称 Vector，是一个有大小也有方向的量，比如 (1,2,3,4) 等。
3. 矩阵，也称 Matrix，是多个向量合并在一起得到的量，比如[(1,2,3),(4,5,6)]等。

为了帮助你更好理解标量、向量和矩阵，我特意准备了一张示意图，你可以结合图片理解。



极客时间

不难发现，几种数据表示其实都是有着联系的，标量可以组合成向量，向量可以组合成矩阵。那么，我们可否将它们看作是一种数据形式呢？

答案是可以的，这种统一的数据形式，在 PyTorch 中我们称之为**张量 (Tensor)**。从标量、向量和矩阵的关系来看，你可能会觉得它们就是不同**“维度”**的 Tensor，这个说法对，也不全对。

说它不全对是因为在 Tensor 的概念中，我们更愿意使用 Rank（秩）来表示这种**“维度”**，比如标量，就是 Rank 为 0 阶的 Tensor；向量就是 Rank 为 1 阶的 Tensor；矩阵就是 Rank 为 2 阶的 Tensor。也有 Rank 大于 2 的 Tensor。当然啦，你如果说维度其实也没什么错误，平时很多人也都这么叫。

说完 Tensor 的含义，我们一起看一下 Tensor 的类型，以及如何创建 Tensor。

Tensor 的类型、创建及转换

在不同的深度学习框架下，Tensor 呈现的特点大同小异，我们使用它的方法也差不多。这节课我们就以 PyTorch 中的使用方法为例进行学习。

Tensor 的类型

在 PyTorch 中，Tensor 支持的数据类型有很多种，这里列举较为常用的几种格式：

Data type	dtype	CPU tensor
32-bit floating point	torch.float32 or torch.float	torch.FloatTensor
64-bit floating point	torch.float64 or torch.double	torch.DoubleTensor
16-bit floating point	torch.float16 or torch.half	torch.HalfTensor
8-bit integer (unsigned)	torch.uint8	torch.ByteTensor
8-bit integer (signed)	torch.int8	torch.CharTensor
16-bit integer (signed)	torch.int16 or torch.short	torch.ShortTensor
32-bit integer (signed)	torch.int32 or torch.int	torch.IntTensor
64-bit integer (signed)	torch.int64 or torch.long	torch.LongTensor
Boolean	torch.bool	torch.BoolTensor

极客时间

一般来说，torch.float32、torch.float64、torch.uint8 和 torch.int64 用得相对较多一些，但是也不是绝对，还是要根据实际情况进行选择。这里你有个印象就行，后面课程用到时我还会进一步讲解。

Tensor 的创建

PyTorch 对于 Tensor 的操作已经非常友好了，你可以通过多种不同的方式创建一个任意形状的 Tensor，而且每种方式都很简便，我们一起来看一下。

直接创建

首先来看直接创建的方法，这也是最简单创建的方法。我们需要用到下面的 torch.tensor 函数直接创建。

复制代码

```
torch.tensor(data, dtype=None, device=None,requires_grad=False)
```

结合代码，我们看看其中的参数是什么含义。

我们从左往右依次来看，首先是 data，也就是我们要传入模型的数据。PyTorch 支持通过 list、 tuple、numpy array、scalar 等多种类型进行数据传入，并转换为 tensor。

接着是 dtype，它声明了你需要返回一个怎样类型的 Tensor，具体类型可以参考前面表格里列举的 Tensor 的 8 种类型。

然后是 device，这个参数指定了数据要返回到的设备，目前暂时不需要关注，缺省即可。

最后一个参数是 requires_grad，用于说明当前量是否需要在计算中保留对应的梯度信息。在 PyTorch 中，只有当一个 Tensor 设置 requires_grad 为 True 的情况下，才会对这个 Tensor 以及由这个 Tensor 计算出来的其他 Tensor 进行求导，然后将导数值存在 Tensor 的 grad 属性中，便于优化器来更新参数。

所以，你需要注意的是，把 requires_grad 设置成 true 或者 false 要灵活处理。**如果是训练过程就要设置为 true，目的是方便求导、更新参数。而到了验证或者测试过程，我们的目的是检查当前模型的泛化能力，那就要把 requires_grad 设置成 Fasle，避免这个参数根据 loss 自动更新。**

从 NumPy 中创建

还记得之前的课程中，我们一同学习了 NumPy 的使用，在实际应用中，我们在处理数据的阶段多使用的是 NumPy，而数据处理好之后想要传入 PyTorch 的深度学习模型中，则需要借助 Tensor，所以 PyTorch 提供了一个从 NumPy 转到 Tensor 的语句：

复制代码

```
torch.from_numpy(ndarry)
```

有时候我们在开发模型的过程中，需要用到一些特定形式的矩阵 Tensor，比如全是 0 的，或者全是 1 的。这时我们就可以用这个方法创建，比如说，先生成一个全是 0 的 NumPy

数组，然后转换成 Tensor。但是这样也挺麻烦的，因为这意味着你要引入更多的包（NumPy），也会使用更多的代码，这会增加出错的可能性。

不过你别担心，PyTorch 内部已经提供了更为简便的方法，我们接着往下看。

创建特殊形式的 Tensor

我们一块来看一下后面的几个常用函数，它们都是在 PyTorch 模型内部使用的。

创建零矩阵 Tensor：零矩阵顾名思义，就是所有的元素都为 0 的矩阵。

```
1 torch.zeros(*size, dtype=None...)
```

复制代码

其中，我们用得比较多的就是 size 参数和 dtype 参数。size 定义输出张量形状的整数序列。

这里你可能注意到了，在函数参数列表中我加入了省略号，这意味着 torch.zeros 的参数有很多。不过。咱们现在是介绍零矩阵的概念，形状相对来说更重要。其他的参数（比如前面提到的 requires_grad 参数）与此无关，现阶段我们暂时不关注。

创建单位矩阵 Tensor：单位矩阵是指主对角线上的元素都为 1 的矩阵。

```
1 torch.eye(size, dtype=None...)
```

复制代码

创建全一矩阵 Tensor：全一矩阵顾名思义，就是所有的元素都为 1 的矩阵。

```
1 torch.ones(size, dtype=None...)
```

复制代码

创建随机矩阵 Tensor：在 PyTorch 中有几种较为经常使用的随机矩阵创建方式，分别如下。

复制代码

```
torch.rand(size)
torch.randn(size)
torch.normal(size, mean, std)
torch.randint(low, high, size)
```

这些方式各自有不同的用法，你可以根据自己的需要灵活使用。

torch.rand 用于生成数据类型为浮点型且维度指定的随机 Tensor，随机生成的浮点数据在 **0~1 区间均匀分布**。

torch.randn 用于生成数据类型为浮点型且维度指定的随机 Tensor，随机生成的浮点数的取值满足**均值为 0、方差为 1 的标准正态分布**。

torch.normal 用于生成数据类型为浮点型且维度指定的随机 Tensor，**可以指定均值和标准差**。

torch.randint 用于生成随机整数的 Tensor，其内部填充的是在[low,high) 均匀生成的随机整数。

Tensor 的转换

在实际项目中，我们接触到的数据类型有很多，比如 Int、list、NumPy 等。为了让数据在各个阶段畅通无阻，不同数据类型与 Tensor 之间的转换就非常重要了。接下来我们一起来看看 int、list、NumPy 是如何与 Tensor 互相转换的。

Int 与 Tensor 的转换：

复制代码

```
a = torch.Tensor(1)
b = a.item()
```

我们通过 torch.Tensor 将一个数字（或者标量）转换为 Tensor，又通过 item() 函数，将 Tensor 转换为数字（标量），item() 函数的作用就是将 Tensor 转换为一个 python number。

list 与 tensor 的转换：

复制代码

```
a = [1, 2, 3]
b = torch.Tensor(a)
c = b.numpy().tolist()
```

在这里对于一个 list a，我们仍旧直接使用 torch.Tensor，就可以将其转换为 Tensor 了。而还原回来的过程要多一步，需要我们先将 Tensor 转为 NumPy 结构，之后再使用 tolist() 函数得到 list。

NumPy 与 Tensor 的转换：

有了前面两个例子，你是否能想到 NumPy 怎么转换为 Tensor 么？对，我们仍旧 torch.Tensor 即可，是不是特别方便。

CPU 与 GPU 的 Tensor 之间的转换：

复制代码

```
CPU->GPU: data.cuda()
GPU->CPU: data.cpu()
```

Tensor 的常用操作

好，刚才我们一起了解了 Tensor 的类型，如何创建 Tensor，以及如何实现 Tensor 和一些常见的数据类型之间的相互转换。其实 Tensor 还有一些比较常用的功能，比如获取形状、维度转换、形状变换以及增减维度，接下来我们一起来看看这些功能。

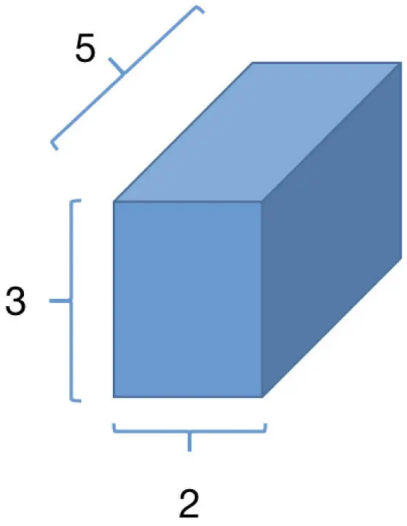
获取形状

在深度学习网络的设计中，我们需要时刻对 Tensor 的情况做到了如指掌，其中就包括获取 Tensor 的形式、形状等。

为了得到 Tensor 的形状，我们可以使用 shape 或 size 来获取。两者的不同之处在于，shape 是 torch.tensor 的一个属性，而 size() 则是一个 torch.tensor 拥有的方法。

复制代码

```
>>> a=torch.zeros(2, 3, 5)
>>> a.shape
torch.Size([2, 3, 5])
>>> a.size()
torch.Size([2, 3, 5])
```



极客时间

知道了 Tensor 的形状，我们就能知道这个 Tensor 所包含的元素的数量了。具体的计算方法就是直接将所有维度的大小相乘，比如上面的 Tensor a 所含有的元素的个数为 2*3*5=30 个。这样似乎有点麻烦，我们在 PyTorch 中可以使用 numel() 函数直接统计元素数量。

复制代码

```
>>> a.numel()
30
```

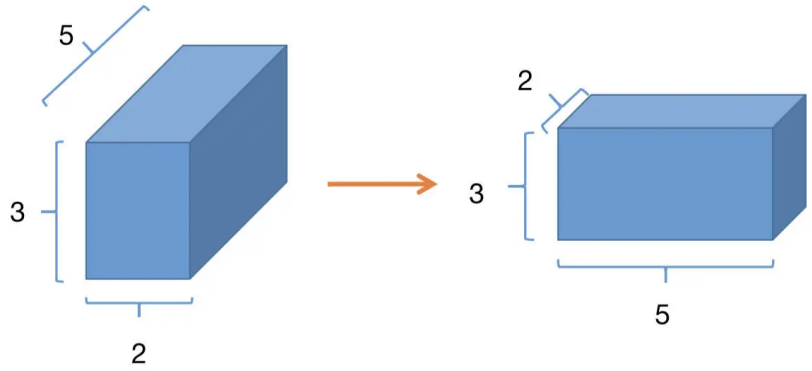
矩阵转秩（维度转换）

在 PyTorch 中有两个函数，分别是 permute() 和 transpose() 可以用来实现矩阵的转秩，或者说交换不同维度的数据。比如在调整卷积层的尺寸、修改 channel 的顺序、变换全连接层的大小的时候，我们就要用到它们。

其中，用 permute 函数可以对任意高维矩阵进行转置，但只有 tensor.permute() 这个调用方式，我们先看一下代码：

复制代码

```
>>> x = torch.rand(2,3,5)
>>> x.shape
torch.Size([2, 3, 5])
>>> x = x.permute(2,1,0)
>>> x.shape
torch.Size([5, 3, 2])
```



极客时间

有没有发现，原来的 Tensor 的形状是[2,3,5]，我们在 permute 中分别写入原来索引位置的新位置，x.permute(2,1,0)，2 表示原来第二个维度现在放在了第零个维度；同理 1 表示原来第一个维度仍旧在第一个维度；0 表示原来第 0 个维度放在了现在的第 2 个维度，形状就变成了[5,3,2]

而另外一个函数 transpose，不同于 permute，它每次只能转换两个维度，或者说交换两个维度的数据。我们还是来看一下代码：

复制代码

```
>>> x.shape
torch.Size([2, 3, 4])
>>> x = x.transpose(1,0)
>>> x.shape
torch.Size([3, 2, 4])
```

需要注意的是，经过了 transpose 或者 permute 处理之后的数据，变得不再连续了，什么意思呢？

还是接着刚才的例子说，我们使用 torch.rand(2,3,4) 得到的 tensor，在内存中是连续的，但是经过 transpose 或者 permute 之后呢，比如 transpose(1,0)，内存虽然没有变化，但是我们得到的数据“看上去”是第 0 和第 1 维的数据发生了交换，现在的第 0 维是原来的第 1 维，所以 Tensor 都会变得不再连续。

那你可能会问了，不连续就不连续呗，好像也没啥影响吧？这么想你就草率了，我们继续来看看 Tensor 的形状变换，学完以后你就知道 Tensor 不连续的后果了。

形状变换

在 PyTorch 中有两种常用的改变形状的函数，分别是 view 和 reshape。我们先来看一下 view。

复制代码

```
>>> x = torch.randn(4, 4)
>>> x.shape
torch.Size([4, 4])
>>> x = x.view(2,8)
>>> x.shape
torch.Size([2, 8])
```

我们先声明了一个[4, 4]大小的 Tensor，然后通过 view 函数，将其修改为[2, 8]形状的 Tensor。我们还是继续刚才的 x，再进行一步操作，代码如下：

复制代码

```
>>> x = x.permute(1,0)
>>> x.shape
torch.Size([8, 2])
>>> x.view(4, 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: view size is not compatible with input tensor's size and stride
```

结合代码可以看到，利用 permute，我们将第 0 和第 1 维度的数据进行了变换，得到了[8, 2]形状的 Tensor，在这个新 Tensor 上进行 view 操作，忽然就报错了，为什么呢？其实就是因为 view 不能处理内存不连续 Tensor 的结构。

那这时候要怎么办呢？我们可以使用另一个函数，reshape：

复制代码

```
>>> x = x.reshape(4, 4)
>>> x.shape
torch.Size([4, 4])
```

这样问题就迎刃而解了。其实 reshape 相当于进行了两步操作，先把 Tensor 在内存中捋顺了，然后再进行 view 操作。

增减维度

有时候我们需要对 Tensor 增加或者删除某些维度，比如删除或者增加图片的几个通道。PyTorch 提供了 squeeze() 和 unsqueeze() 函数解决这个问题。

我们先来看 squeeze()。如果 dim 指定的维度的值为 1，则将该维度删除，若指定的维度值不为 1，则返回原来的 Tensor。为了方便你理解，我还是结合例子来讲解。

复制代码

```
>>> x = torch.rand(2,1,3)
>>> x.shape
torch.Size([2, 1, 3])
>>> y = x.squeeze(1)
>>> y.shape
torch.Size([2, 3])
>>> z = y.squeeze(1)
>>> z.shape
torch.Size([2, 3])
```

结合代码我们可以看到，我们新建了一个维度为[2, 1, 3]的 Tensor，然后将第 1 维度的数据删除，得到 y，squeeze 执行成功是因为第 1 维度的大小为 1。然而在 y 上我们打算进一步删除第 1 维度的时候，就会发现删除失败了，这是因为 y 此刻的第 1 维度的大小为 3，suqeeze 不能删除。

unsqueeze()：这个函数主要是对数据维度进行扩充。给指定位置加上维数为 1 的维度，我们同样结合代码例子来看看。

复制代码

```
>>> x = torch.rand(2,1,3)
>>> y = x.unsqueeze(2)
>>> y.shape
torch.Size([2, 1, 1, 3])
```

这里我们新建了一个维度为[2, 1, 3]的 Tensor，然后在第 2 维度插入一个数据，这样就得到了一个[2,1,1,3]大小的 tensor。

小结

之前我们学习了 NumPy 相关的操作，如果把 NumPy 和 Tensor 做对比，就不难发现它们之间有很多共通的内容，共性就是两者都是数据的表示形式，都可以看作是科学计算的通用工具。但是 NumPy 和 Tensor 的用途是不一样的，NumPy 不能用于 GPU 加速，Tensor 则可以。

这节课我们一同学习了 Tensor 的创建、类型、转换、变换等常用功能，通过这几个功能，我们就可以对 Tensor 进行最基本也是最常用的操作，这些都是必须要牢记的内容。

此外，在实际上，真正的项目实战中还有个非常多的操作种类，其中较为重要的是**数学计算操作**，比如加减乘除、合并、连接等。但是这些操作如果一个一个列举出来，数量极其繁多，你也会感觉很枯燥，所以在后续的课程中，咱们会在具体的实战环节来学习相关的数学操作。

下一节课的内容，咱们会对 Tensor 的变形、切分等高级操作进行学习，这是一个很好玩儿的内容，敬请期待。

每课一练

在 PyTorch 中，有 torch.Tensor() 和 torch.tensor() 两种函数，它们的区别是什么呢？

欢迎你在留言区和我交流，也推荐你把今天的内容分享给更多同事和朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | NumPy（下）：深度学习中的常用操作

精选留言（1）

写留言

Joe Black

2021-10-18

torch.Tensor()是Python类，是默认张量类型torch.FloatTensor()的别名，而torch.tensor()是Python函数。

展开